Кулиев Фёдор Русланович, группа 20-1

Лабораторная работа № 1

**Задание 1.1.**

**Цель работы**

Осуществить обучение нейрона выполнению функции логического «ИЛИ».

2. Осуществить попытку обучения одного нейрона выполнению функции исключающего «ИЛИ» и посмотреть на результат.

3. Построить и обучить модель для решения функции исключающего «ИЛИ» (XOR) (используйте adam оптимизатор для более быстрого обучения).

4. Построить график активационной функции заданного вида (по вариантам, по номеру в списке).

**Задание**

1. Осуществить обучение нейрона выполнению функции логического «ИЛИ».

2. Осуществить попытку обучения одного нейрона выполнению функции исключающего «ИЛИ» и посмотреть на результат.

3. Построить и обучить модель для решения функции исключающего «ИЛИ» (XOR) (используйте adam оптимизатор для более быстрого обучения).

4. Построить график активационной функции заданного вида (по вариантам, по номеру в списке).

**Код программы / ответ на вопрос и т.п.**

# Подключение необходимых библиотек

import numpy as np

import matplotlib.pyplot as plt

import torch

import torch.nn as nn

import torch.nn.functional as F

from torch.optim import Adam

from tqdm import tqdm

# Данные для обучения нейронной сети

x = torch.tensor([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=torch.float32)

y\_or = torch.tensor([[0], [1], [1], [1]], dtype=torch.float32)

y\_xor = torch.tensor([[0], [1], [1], [0]], dtype=torch.float32)

# Нейронная сеть для решения задачи логического или

class model\_OR(nn.Module):

    def \_\_init\_\_(self):

        super(model\_OR, self).\_\_init\_\_()

        self.fc1 = nn.Linear(2, 1)

    def forward(self, x):

        x = F.sigmoid(self.fc1(x))

        return x

# Параметры модели

model\_or = model\_OR()

optimizer = Adam(params=model\_or.parameters(), lr=1e-3)

loss\_func = nn.BCELoss()

# Процесс обучения модели

losses = []

for epoch in tqdm(range(5000)):

    outputs = model\_or(x)

    loss = loss\_func(outputs, y\_or)

    loss.backward()

    losses.append(loss.item())

    optimizer.step()

    optimizer.zero\_grad()

# Попробуем решить задачу исключающего ИЛИ однослойной нейронной сетью

class model\_XOR1(nn.Module):

    def \_\_init\_\_(self):

        super(model\_XOR1, self).\_\_init\_\_()

        self.fc1 = nn.Linear(2, 1)

    def forward(self, x):

        # Преобразуем входной тензор в одномерный вектор

        x = F.sigmoid(self.fc1(x))

        return x

# Параметры модели

model\_xor1 = model\_XOR1()

optimizer\_xor1 = Adam(params=model\_xor1.parameters(), lr=1e-3)

loss\_func = nn.BCELoss()

# Процесс обучения модели

losses = []

for epoch in tqdm(range(5000)):

    outputs = model\_xor1(x)

    loss = loss\_func(outputs, y\_xor)

    loss.backward()

    losses.append(loss.item())

    optimizer\_xor1.step()

    optimizer\_xor1.zero\_grad()

# Создаём многослойную нейросеть для решения задачи логического ИЛИ

class model\_XOR2(nn.Module):

    def \_\_init\_\_(self):

        super(model\_XOR2, self).\_\_init\_\_()

        self.fc1 = nn.Linear(2, 2)

        self.fc2 = nn.Linear(2, 1)

    def forward(self, x):

        x = F.sigmoid(self.fc1(x))

        x = F.sigmoid(self.fc2(x))

        return x

# Параметры модели

model\_xor2 = model\_XOR2()

optimizer\_xor2 = Adam(params=model\_xor2.parameters(), lr=1e-3)

loss\_func = nn.BCELoss()

# Процесс обучения модели

losses = []

for epoch in tqdm(range(5000)):

    outputs = model\_xor2(x)

    loss = loss\_func(outputs, y\_xor)

    loss.backward()

    losses.append(loss.item())

    optimizer\_xor2.step()

    optimizer\_xor2.zero\_grad()

# Функции активации

def sigmoid(input):

  return 1 / (1 + np.exp(-input))

def tanh(input):

  return (np.exp(2\*input) - 1) / (np.exp(2\*input) + 1)

def relu(input):

  return (max(0, input))

**Результаты выполнения**

print("OR")

print("x\ty\toutput")

for i in range(4):

    print(f"{x[i].detach().numpy()}\t{y\_or[i].detach().numpy()}\t{round(model\_or(x[i]).item())}")

OR

x y output

[0. 0.] [0.] 0

[0. 1.] [1.] 1

[1. 0.] [1.] 1

[1. 1.] [1.] 1

print("XOR")

print("x\ty\toutput")

for i in range(4):

    print(f"{x[i].detach().numpy()}\t{y\_xor[i].detach().numpy()}\t{round(model\_xor1(x[i]).item())}")

XOR

x y output

[0. 0.] [0.] 1

[0. 1.] [1.] 1

[1. 0.] [1.] 1

[1. 1.] [0.] 0

print("XOR")

print("x\ty\toutput")

for i in range(4):

    print(f"{x[i].detach().numpy()}\t{y\_xor[i].detach().numpy()}\t{round(model\_xor2(x[i]).item())}")

XOR

x y output

[0. 0.] [0.] 0

[0. 1.] [1.] 1

[1. 0.] [1.] 1

[1. 1.] [0.] 0

# Вывод на экран графиков функций активации

act\_x = np.linspace(-10, 10, 101)

sigmoid\_y = [sigmoid(x) for x in act\_x]

tanh\_y = [tanh(x) for x in act\_x]

relu\_y = [relu(x) for x in act\_x]

plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1)

plt.plot(act\_x, sigmoid\_y)

plt.title("Sigmoid")

plt.subplot(1, 3, 2)

plt.plot(act\_x, tanh\_y)

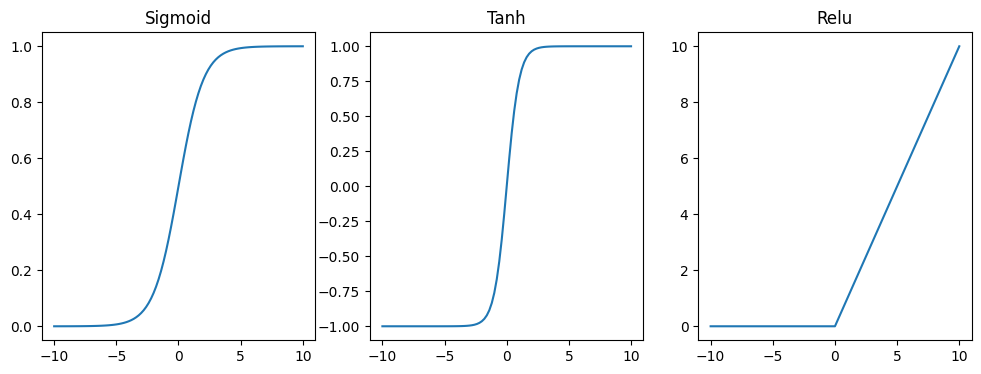
plt.title("Tanh")

plt.subplot(1, 3, 3)

plt.plot(act\_x, relu\_y)

plt.title("Relu")

plt.show()



**Выводы**

Целью работы было изучение и решение задачи логического ИЛИ с помощью нейронных сетей. Были построены и обучены 3 модели нейронных сетей с различными функциями активации. Были получены и проанализированы графики функций активации для каждой модели.

**Задание 1.2.**

**Цель работы**

Создать и обучить нейронную сеть для распознавания рукописных цифр.

**Задание**

1. Попробуйте протестировать обученную нейронную сеть на ваших личных цифрах нарисованных, например, в Paint (на чёрном фоне, белым цветом).
2. Попробуйте подать сети цифру, немного сдвинутую вправо или влево от центра, сможет ли сеть распознать вашу цифру?
3. Попробуйте подать сети перевёрнутую цифру, сможет ли сеть распознать вашу цифру?

**Код программы / ответ на вопрос и т.п.**

# Подключение необходимых библиотек

import torch

import torch.nn as nn

from torch.optim import Adam

from torch.utils.data import Dataset, DataLoader

from torchvision import datasets

from torchvision import transforms

import matplotlib.pyplot as plt

import torch.nn.functional as F

from tqdm import tqdm

# Переключение работы нейросети на GPU, если это возможно

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

print(f"Using {device} device")

# Загрузка обучающей и тестовой выборки

train\_ds = datasets.MNIST(

    root="MNIST\_data",

    train=True,

    download=True,

    transform=transforms.ToTensor(),

)

test\_ds = datasets.MNIST(

    root="MNIST\_data",

    train=False,

    download=True,

    transform=transforms.ToTensor(),

)

train\_dataloader = DataLoader(train\_ds, batch\_size=256, shuffle=True, drop\_last=True)

test\_dataloader = DataLoader(test\_ds, batch\_size=256, shuffle=True, drop\_last=True)

# Визуализация изображений и описаний

train\_features, train\_labels = next(iter(train\_dataloader))

print(f"Feature batch shape: {train\_features.size()}")

print(f"Labels batch shape: {train\_labels.size()}")

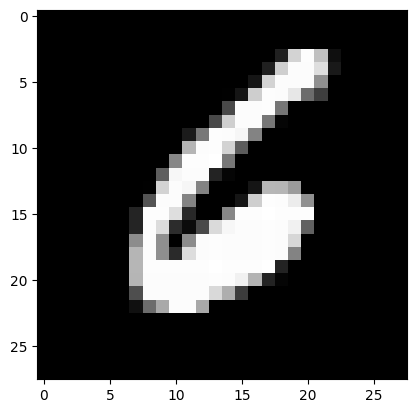
img = train\_features[0].squeeze()

label = train\_labels[0]

plt.imshow(img, cmap="gray")

plt.show()

print(f"Label: {label}")



Label: 6

# Модель нейросети для распознавания рукописных цифр

class NN(nn.Module):

    def \_\_init\_\_(self):

        super(NN, self).\_\_init\_\_()

        self.fc1 = nn.Linear(28\*28, 256)

        self.fc2 = nn.Linear(256, 128)

        self.fc3 = nn.Linear(128, 10)

    def forward(self, x):

        x = x.view(-1, 28 \* 28)

        x = F.relu(self.fc1(x))

        x = F.relu(self.fc2(x))

        x = F.softmax(self.fc3(x), dim=1)

        return x

# Объявление модели и её параметров обучения

model = NN()

model.to(device)

optimizer = Adam(params=model.parameters(), lr=1e-3)

loss\_func = nn.CrossEntropyLoss()

# Процесс обучения модели

num\_iter = 10

print('Train')

train\_loss, train\_acc = [], []

for i in tqdm(range(num\_iter)):

    counter = 0

    train\_running\_loss = 0.0

    train\_running\_correct = 0

    for data, label in train\_dataloader:

        data = data.to(device)

        label = label.to(device)

        outputs = model(data)

        loss = loss\_func(outputs, label)

        train\_running\_loss += loss.item()

        \_, preds = torch.max(outputs.data, 1)

        train\_running\_correct += (preds == label).sum().item()

        loss.backward()

        optimizer.step()

        optimizer.zero\_grad()

        counter += 1

    train\_loss.append(train\_running\_loss / counter)

    train\_acc.append(100. \* (train\_running\_correct / len(train\_dataloader.dataset)))

Train

100%|██████████| 10/10 [00:42<00:00, 4.24s/it]

plt.figure(figsize=(10, 7))

plt.plot(

    train\_loss, linestyle='-',

    label='train loss'

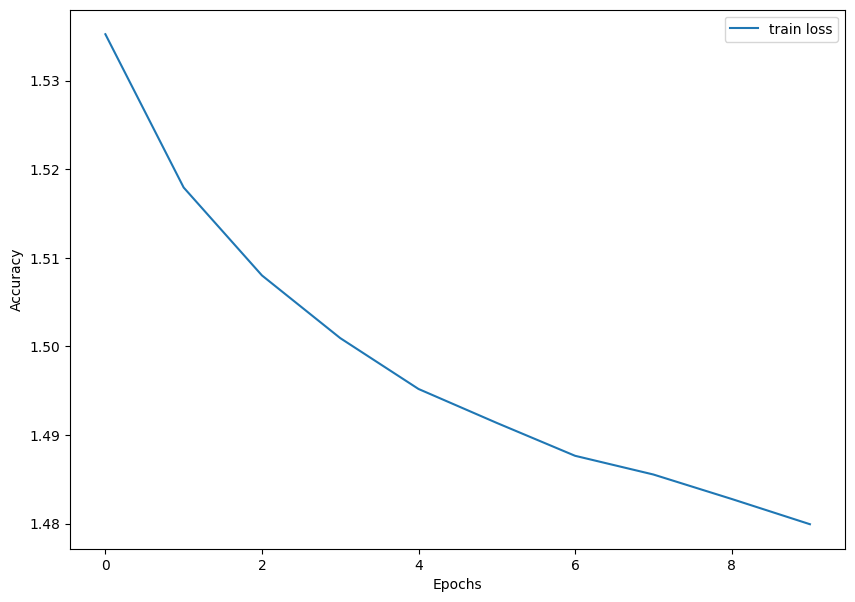
)

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()



plt.figure(figsize=(10, 7))

plt.plot(

    train\_acc, linestyle='-',

    label='train accuracy'

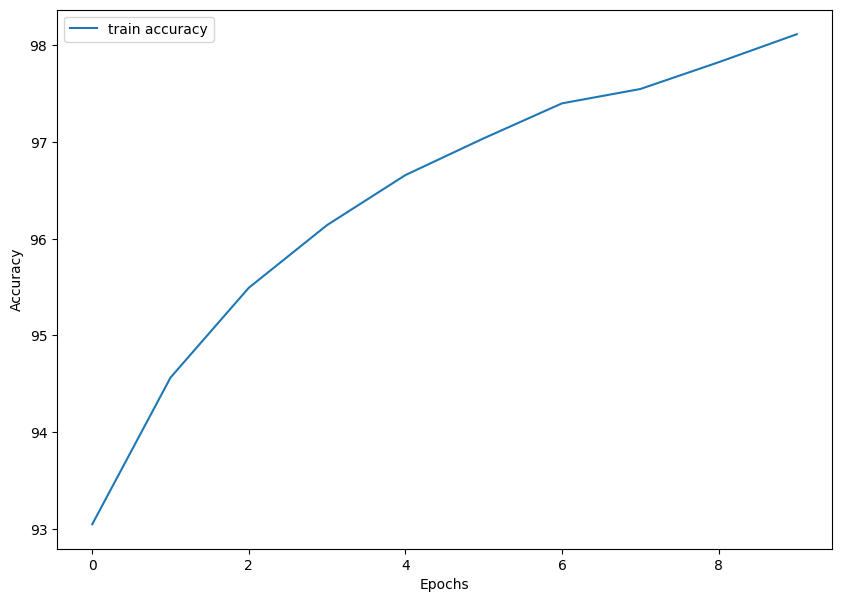
)

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()



Результаты выполнения

# Тест модели

test\_features, test\_labels = next(iter(test\_dataloader))

img = test\_features[0].squeeze()

label = test\_labels[0]

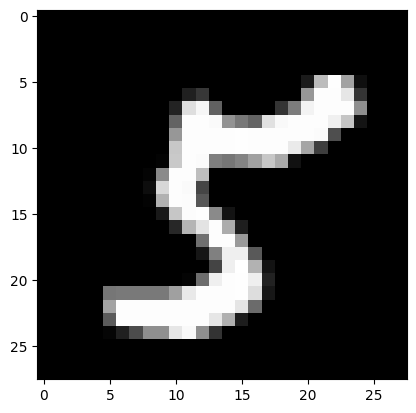
pred = model(test\_features[0].to(device)).argmax()

plt.imshow(img, cmap="gray")

plt.show()

print(f"Label: {label}")

print(f"Pred: {pred}")



Label: 5

Pred: 5

# Подсчёт точности модели

correct = 0

total = len(test\_dataloader)

for data, label in test\_dataloader:

data = data.to(device)

output = model(data).argmax().cpu()

if (output == label):

correct += 1

print('Accuracy of the network on the test images: %d %%' % (100 \* correct / total))

Accuracy of the network on the test images: 94 %

1. Проверка работы на собственном изображении

# Open your image from file

from PIL import Image

path = "Image1.jpg"

img = Image.open(path)

img\_tensor = transforms.ToTensor()(img)

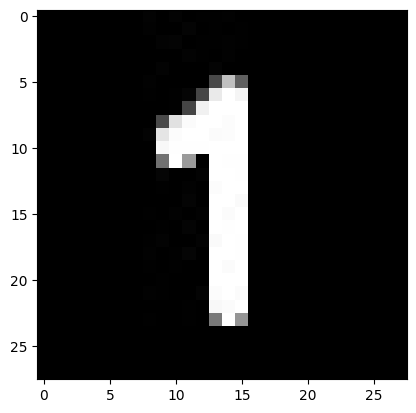
print(f"Predicted value = {model(img\_tensor[0].to(device)).argmax()}")

print("Your image:")

plt.imshow(img\_tensor[0].detach().numpy(), cmap="gray")

Predicted value = 1

Your image:



1. Проверка работы на собственном изображении, сдвинутом в сторону

# Проверка работы нейронной сети на сдвинутом изображении

path = "Image2.jpg"

img = Image.open(path)

img\_tensor = transforms.ToTensor()(img)

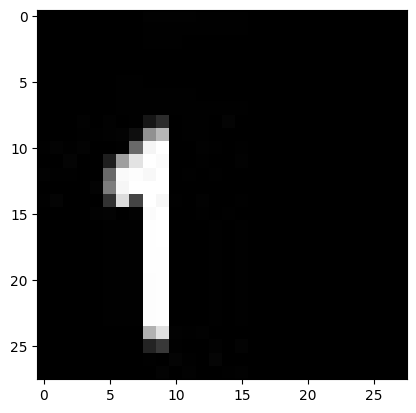
print(f"Predicted value = {model(img\_tensor[0].to(device)).argmax()}")

print("Your image:")

plt.imshow(img\_tensor[0].detach().numpy(), cmap="gray")

Predicted value = 0

Your image:



1. Проверка работы на перевёрнутом изображении

path = "Image3.jpg"

img = Image.open(path)

img\_tensor = transforms.ToTensor()(img)

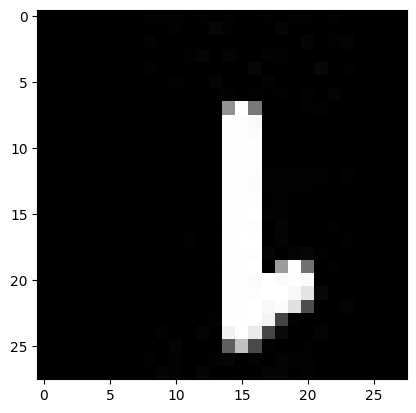
print(f"Predicted value = {model(img\_tensor[0].to(device)).argmax()}")

print("Your image:")

plt.imshow(img\_tensor[0].detach().numpy(), cmap="gray")

Predicted value = 1

Your image:



**Выводы**

В результате выполнения работы, была модель нейронной сети, которая способна распознавать рукописные цифры. Точность модели равна 94%. Данная нейросеть не смогла распознать сдвинутое в сторону изображение цифры, но смогла распознать перевёрнутое. Скорее всего, изображение было распознано из-за того, что цифра 1 похожа на себя, даже если её перевернуть. Данная нейросеть не способна различать различные признаки изображения, как свёрточная сеть, поэтому она не способна распознавать сдвинутые или перевёрнутые изображения.